Week 2: Selection Structures, Functions, List, Plotting

Professor Aaron Quigley

Thanks to Chun Tung Chou

and Ashesh Mahidadia

The Story So Far

```python
# The temperature in Fahrenheit to be converted
temp_fahrenheit = 80 # Change here if needed

# Convert to Celsius using standard formula
temp_celsius = (temp_fahrenheit - MELTING_POINT_FAHRENHEIT) * RATIO

# Output the temperature in Celsius
print(temp_fahrenheit,'in F = ',temp_celsius,'in C')
```

- Write and run programs in the Spyder editor
- Variables: numbers and string
- Assignment, data types, print(), input()

- A program contains a sequence of instructions which are executed sequentially

This week's topics

- Selection structure ⬅
- Functions
- List
- Plotting

Control structure

- Your computer program may need to take different actions depending on the situation
- This is achieved via control structure
- Python has a few different types of control structure
- We will look at selection structure this week
- Selection structure is similar to making decisions
  - There are plenty of real-life examples …

Examples of control

If too hot
    increase cold air flow
Otherwise too cold
    decrease cold air flow

If frontal collision detected
    trigger the air bag

Modern luxury vehicles contain many computer programs (about 100 million lines of code)

http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code

Control is used extensively in engineering:

- Vehicle stability control
- Precision manufacturing
- Chemical process control
- and more

Fault tolerant control

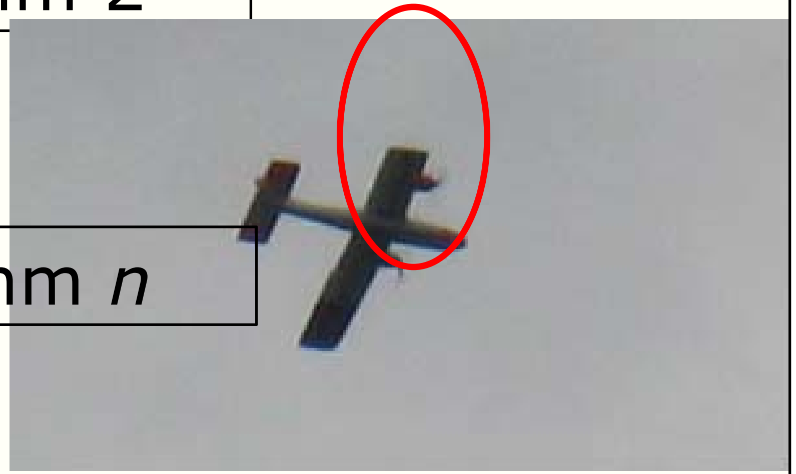
- Example: How to keep the flight going when the plane has experienced severe damages?
- Automatic fault diagnosis and control

Normal → Normal control

Damage 1 → Algorithm 1

Damage 2 → Algorithm 2

Damage *n* → Algorithm *n*



Chowdhary et al. Autonomous Guidance and Control of Airplanes under Severe Damage, 2011

Boolean expression

- Selection structure is based on whether a condition (or a set of conditions) is true or false

- Boolean expression
  - A statement that is either true or false

- In English language, the following are Boolean expressions:
  - UNSW is a university in New Zealand (False)
  - The number 3 is greater than or equal to 3 (True)
  - The number 5 is less than 7 and greater than 11 (False)

Boolean expressions in Python

- Try the following in the console

```
In [3]: 2 > 3 #Is 2 greater than 3?
Out[3]: False

In [4]: 3 >= 3 # Is 3 greater than or
equal to 3?
Out[4]: True
```

Relational operators in Python

Relation	Python operator
Greater than	>
Greater than or equal to	>=
Less than	<
Less than or equal to	<=
Equal	==
Not equal to	!=

```
In [12]: a = 17; b = 18;

In [13]: a == a
In [14]: a != b

In [15]: a + 1 == b
```

- You can compare expressions
- Precedence rule
  - Computation before comparison

Boolean variables

- In Python, Boolean variables can take the value of True or False
  - True and False are Python keywords
    - Note: First letter is capital
    - Remember: You can't use keywords as variable names

```
In [35]: c = 4; var1 = c + 1 == 5

In [36]: var1
Out[36]: True

In [37]: var2 = True

In [38]: var3 = False
```

Computation, then comparison, then assignment
var1 = ((c+1) == 5)

You can directly assign True or False to variables

var1, var2 and var3 are Boolean variables. Try type(var1)

Selection using if/else

- We will write a Python program that
  - Asks the user to input a number
  - Determines if the number is non-negative or negative
    - Non-negative means zero or positive
  - Prints the appropriate output to the user

- We have written the first part of the program in ifelse_prelim.py. The first part
  - Asks the user to input a number

- The file is on the course website. Download and open the file. You will type the rest of the code in.

Selection using if/else

- Type lines 20-23 as shown below
  - Don't forget the colon at the end of lines 20 and 22
  - You can cut-and-paste the print statements in lines 25 and 26 to save typing

```
19 # Decide if number is non-negative or negative
20 if num >= 0:
21     print('The number entered is non-negative')
22 else:
23     print('The number entered is negative')
```

The 4 spaces (= 1 tab) on lines 21 and 23 are called indentation. They are part of Python syntax.

The if part

The statement in the if part is executed if the **condition** is evaluated to be **True**

```python
# Decide if number is non-negative or negative
if num >= 0:
    print('The number entered is non-negative')
else:
    print('The number entered is negative')
```

```
Please input a number: 2.9
The number entered is non-negative
```

The else part

The statement in the else part is executed if the **condition** is evaluated to be **False**

```python
# Decide if number is non-negative or negative
if num >= 0:
    print('The number entered is non-negative')
else:
    print('The number entered is negative')
```

Logic reasoning that you need to know:

(num >= 0) is **False** implies that (num < 0)

```
Please input a number: -3.7
The number entered is negative
```

Selection `if/else`

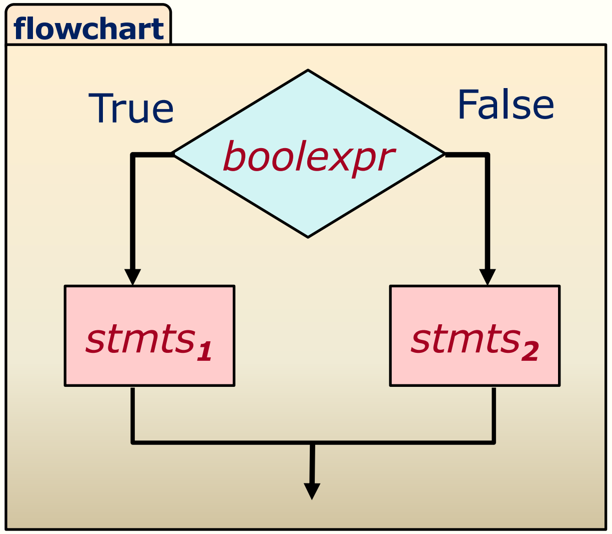
- The `if/else` statement is used to make decisions using Boolean expressions

- Symmetric form:

```
if boolean-expression:
    statement-list1
else:
    statement-list2
```

indented! → *statement-list1*

indented! → *statement-list2*



boolean-expression is evaluated

- If it evaluates to `True`, *statement-list1* is executed
- Otherwise, *statement-list2* is executed

Quiz

- You want to develop a program that does the following
  - Given two float variables *a* and *b*, the variable *bigger_of_two_numbers* should be assigned the bigger value of *a* and *b*

- Most of the code has been written in the file bigger_prelim.py

- What Boolean expression should be put in Line 20?

```
16 # Input two numbers
17 a = float(input('Please input the 1st number:'))
18 b = float(input('Please input the 2nd number:'))
19
20 if    :
21     bigger_of_two_numbers = a
22 else:
23     bigger_of_two_numbers = b
24
25 print('The bigger number is',bigger_of_two_numbers)
```

Program testing

- You write computer programs with an intention for the programs to do certain tasks

- If a program runs without error, it doesn't mean it works in the way you intend it to be
  - Language analogy:
    - A correct program is similar to a grammatically correct sentence
    - However, a grammatically correct sentence doesn't mean it's a sentence that makes sense

- You want to test the programs to ensure they function as intended

- A way to test a program is to give it many different sets of input and check whether you get the expected output for each set of inputs

Quiz

- You have written a program which returns the bigger value of 2 numbers

- Which of the following set is a better choice to test this program?

Set 1	Set 2
a = 5; b = 2	a = 5; b = 2
a = -5; b = -10	a = 5; b = 10

- Hint: You want to cover all possibilities

You said we should cover all possibilities …

- Should we try?

```
a = 5
b = “HaHaHaGotYou”
```

- Yes! Definitely!
- Although you intend the users to enter only numbers, but a defensive programmer ensures that a piece of software continues to function under unexpected circumstances
- This is hard but no one likes software that crashes
- We will come back on this later on in the course

You can have multiple statements in the if or else section

These statements in the if part are executed if the **condition** is evaluated to be **True**

```
# Decide if number is non-negative or negative
if num >= 0:
    print('The number entered is non-negative')
    abs_num = num
else:
    print('The number entered is negative')
    abs_num = -num
```

Important: **These statements must have the same indentation**

Indentation tells where the else block ends

```
16 # Ask the user to input a number
17 num = float(input('Please input a number: '))
18
19 # Decide if number is non-negative or negative
20 if num >= 0:
21     print('The number entered is non-negative')
22     abs_num = num
23 else:
24     print('The number entered is negative')
25     abs_num = -num
26
27 print('The absolute value of ',num,' is ',abs_num)
```

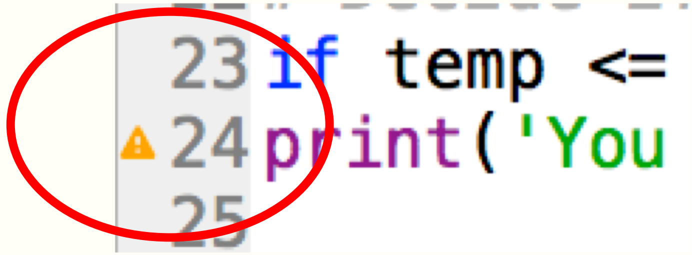
This statement is **outside** of if/else because it is **not** indented. It is executed no matter whether (num >= 0) is **True** or **False.**

Indentation tells whether a statement is inside or outside of if/else

Let us add Line 22, 25 and 27 to ifelse_prelim.py and run the code

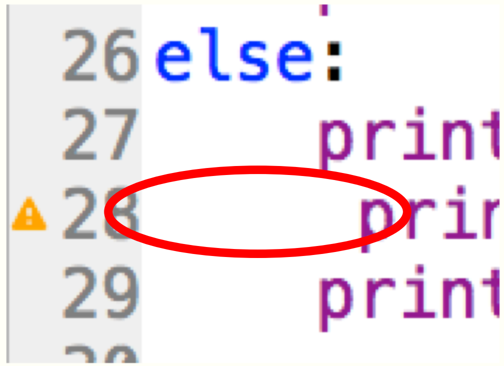
Indentation error

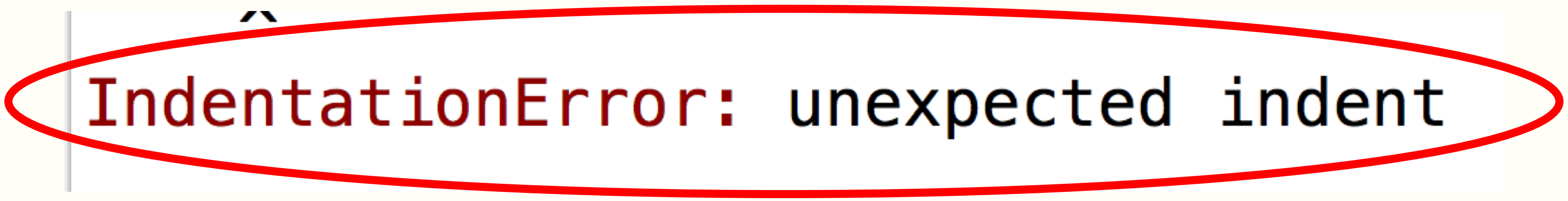
Problem: No indentation.



Problem: Irregular indentation.

Note: The editor warns you.





Program development tips

- The programs that we have written is still short, in terms of the number of lines of code
- You will be writing longer programs
- It's a good time to start learning some program development tips

- Incremental development
  - Break the program into sections
    - Code a section
    - Test that it is working
    - Then move onto the next section

Incremental development example

```
# Ask the user to input a number
num = float(input('Please input a number: '))

# Decide if number is non-negative or negative
if num >= 0:
    print('The number entered is non-negative')
    abs_num = num
else:
    print('The number entered is negative')
    abs_num = -num

print('The absolute value of ',num,' is ',abs_num)
```

- The above shows the code for the if/else example
- You can develop the four sections one after another

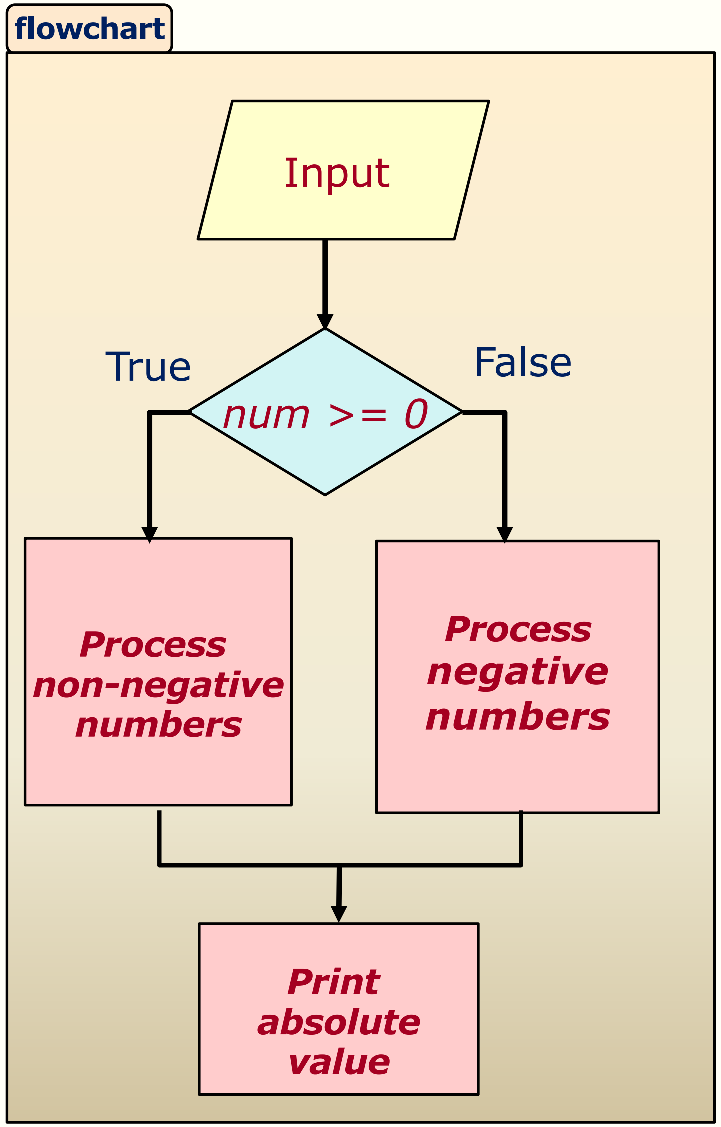
Program planning

- In school, you learn to use storyboards to plan your story before actually writing it out
- The philosophy of **separating** planning and writing also applies to programming
  - First plan the workflow in your program
  - After planning, write the code
- There are two common planning aids:
  - Flowcharts
  - Pseudocode

Flowcharts

- Flowcharts are used to visualise workflow of a computer program
  - Diamond block for decision
  - Rectangular block for process
  - Parallelogram for input

- Example: The flowchart on the right corresponds to the program 2 slides earlier



Pseudocode

- Pseudocode shows the structure of the program but is written in human language
- For example:

```
Get a number from the user

IF the number is greater than or equal to 0 THEN
        Print the number is non-negative
ELSE (Note for myself: this means the number is neg)
        Print the number is negative
END OF IF
```

Always plan first

- You can use any planning aid you prefer but
  - Always plan first
  - Then write your code

Boolean operators

- You can combine Boolean expressions by using Boolean operators
- There are three Boolean operators

  binary **and** **or**

  unary **not**
- Examples:

  (a > 5) and (b > 10)

  (a >= 5) or (b != 10)
- Parentheses have been added for readability but you don't need them because **and** as well as **or** have lower precedence than the comparison operators

  a > 5 and b > 10

  a >= 5 or b != 10

Example code

```python
# Input two numbers
a = float(input('Please input the 1st number:'))
b = float(input('Please input the 2nd number:'))

if (a > 5) and (b > 10):  # Parentheses not needed
    print('The condition is True')
else:
    print('The condition is False')
```

- Code in boolean.py
- Let's try **or** as well as **not**
- Quiz: What is **not**(b > 10) equivalent to?

Truth Tables

Truth tables establish meaning of operators by enumerating each combination of operands and showing what the operation yields

Notation: T = True, F = False

A	B	A and B	A or B	not A
F	F	F	F	T
F	T	F	T	T
T	F	F	T	F
T	T	T	T	F

and – both True or – either True

not – complement

Examples

- You want to test whether the variable x is within the interval [0.0,1.0)

```
x >= 0.0 and x < 1.0
```

- When is the following true? Can you replace it by a single comparison?

```
j < 0 or j > 0
```

- When is the following true? What can you use it for?

```
a == b and b == c
```

De Morgan’s Laws

- De Morgan’s Laws are important because they help us to make logical reasoning. There are two forms.

```
not (E1 and E2) is equivalent to (not E1) or (not E2)

not (E1 or E2)  is equivalent to (not E1) and (not E2)
```

- You will find it useful when we learn to code using the while-statement in a number of weeks’ time

- For example: In a game, if a player has got 15 points or more, then the game ends

De Morgan’s Laws: Example

- The game ends if

```
player_1_score >= 15 or player_2_score >= 15
```

- The game continues if

```
not(player_1_score >= 15 or player_2_score >= 15)
```

- By De Morgan’s Law, this is equivalent to:

```
not(player_1_score >= 15) and not(player_2_score >= 15)
```

```
player_1_score < 15 and player_2_score < 15
```

Remarks

- Sometimes, you need to be explicit.
  - In English, you say x is not equal to 6, 7 or 8
  - In Python (and many programming languages), you need to write

```
(x != 6) or (x != 7) or (x != 8)
```

- Python allows you to be implicit with and, e.g. the expressions on the left can be shortened to those on their right

```
x >= 0.0 and x < 1.0        0.0 <= x < 1.0
```

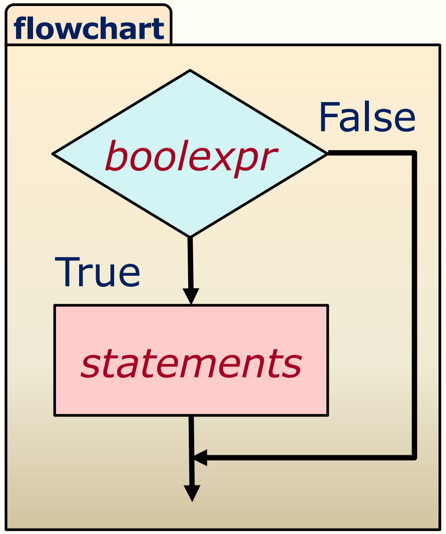
```
a == b and b == c           a == b == c
```

Selection – `if`

- Simpler form:

```
if boolean-expression:
    statements
```

indented! → statements



- *boolean-expression* is evaluated
  - If it evaluates to `True`, statements are executed
  - otherwise (i.e., it must be `False`) skip over statements and continue with rest of program
- Also referred to as a *conditional statement*

Example using if

- Open the file if_demo.py

```
# Ask the user to input a number
x = float(input('Please input a number: '))

# Selection
if x > 0:
    print('The number entered is positive')

print('The cube of the input is ',x**3)

```

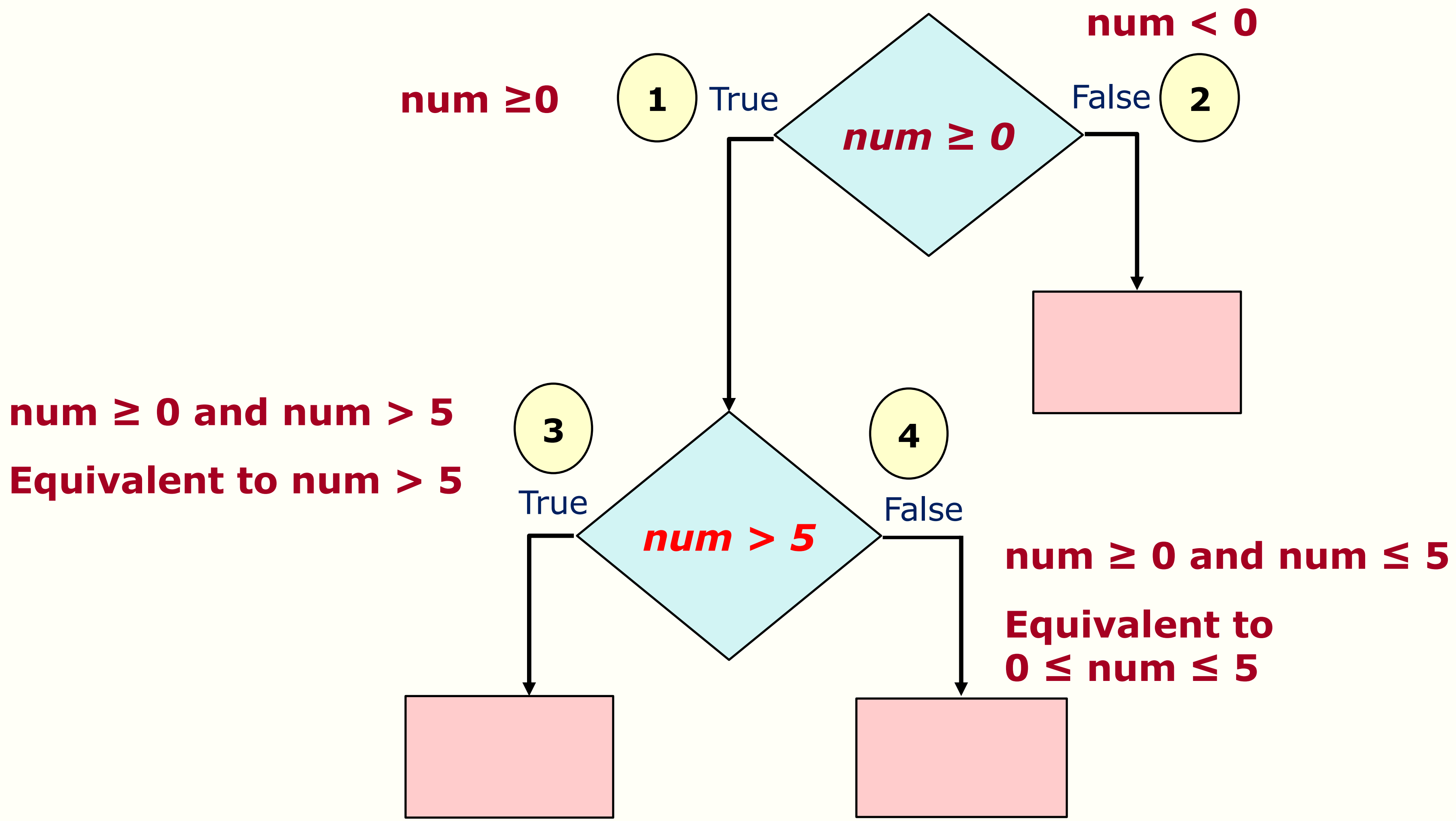
Nested if

- You can have a if/else inside another if/else
- For example: (nested_if.py)

```python
if num >= 0:
    if num > 5:
        print('The number entered is >5')
    else:
        print('The number entered is in the interval [0,5]')
else:
    print('The number entered is negative')

```

```python
if num >= 0:
    if num > 5:
        print('The number entered is >5')
    else:
        print('The number entered is in the interval [0,5]')
else:
    print('The number entered is negative')
```

3 4 2



Quiz

```
if boolean_exp_1:
    if boolean_exp_2:
        statement_list_1
    else:
        statement_list_2
else:
    if boolean_exp_3:
        statement_list_3
    else:
        statement_list_4
```

- Under what condition will statement_list_2 be executed? Why?
  a. boolean_exp_1 is true and boolean_exp_2 is true
  b. boolean_exp_1 is true and boolean_exp_2 is false
  c. boolean_exp_1 is false and boolean_exp_2 is true
  d. boolean_exp_1 is false and boolean_exp_2 is false

More complex form

Chained form, symmetric form generalised to *n*:

```
if Boolean_expression1:
    statement_list1
elif Boolean_expression2:
    statement_list2
elif Boolean_expression3:
    statement_list3
            …
else:
    statement_list_n
```

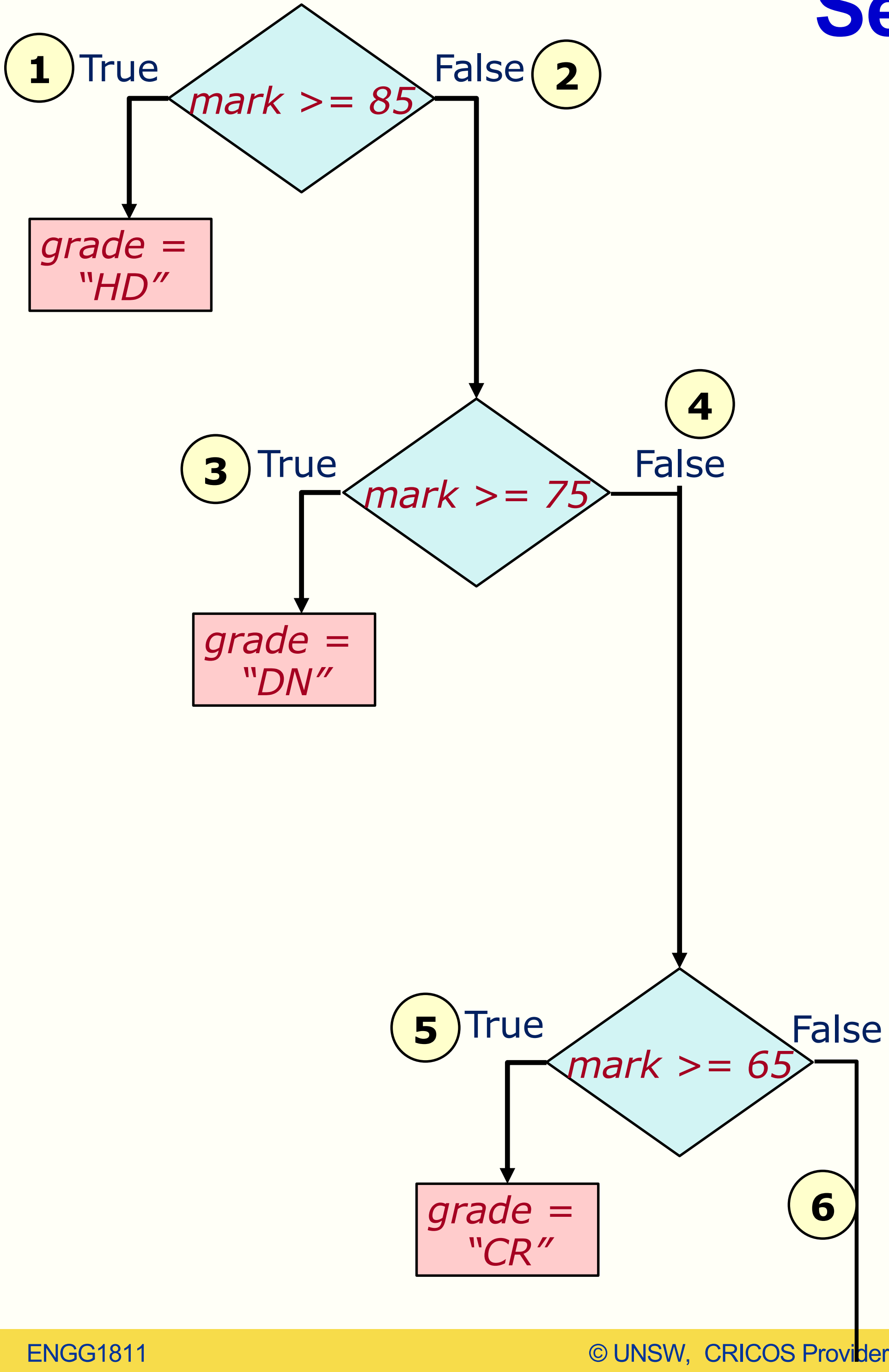
Chained Selection example: Classification

Often need to classify a value based on ranges, such as deriving UNSW grade from mark:

```
# Precondition (assumption): 0 <= mark <= 100
if mark >= 85:
    grade = "HD"
elif mark >= 75:   # (mark >= 85 is false) and (mark >= 75 is true)
    grade = "DN"
elif mark >= 65:
    grade = "CR"
elif mark >= 50:
    grade = "PS"
else:            # Not (mark >= 50), so mark < 50
    grade = "FL"
```

Code in mark2grade.py

Selection `If/elif/else`



1. mark >= 85
2. mark < 85
3. mark < 85 and mark >=75
4. mark < 85 and not(mark >= 75)

   ≣ mark < 85 and mark < 75

   ≣ mark < 75
5. mark < 75 and mark >=65
6. mark < 65

Quiz

Will the following code work? Changes are inside the dashed box. Why?

↓What we had earlier

```
if mark >= 85:
    grade = "HD"
elif mark >= 75:
    grade = "DN"
elif mark >= 65:
    grade = "CR"
elif mark >= 50:
    grade = "PS"
else:
    grade = "FL"
```

```
if mark >= 85:
    grade = "HD"
elif mark >= 75:
    grade = "DN"
elif mark >= 50:
    grade = ”PS"
elif mark >= 65:
    grade = ”CR"
else:
    grade = "FL"
```

Remark

The following code works but the part within the orange box is redundant.

```
if mark >= 85:
    grade = "HD"
elif mark < 85 and mark >= 75:
    grade = "DN"
```

Should be written as:

```
if mark >= 85:
    grade = "HD"
elif mark >= 75:
    grade = "DN"
```

This week's topics

- Selection structure
- Functions
- List
- Plotting

Functions

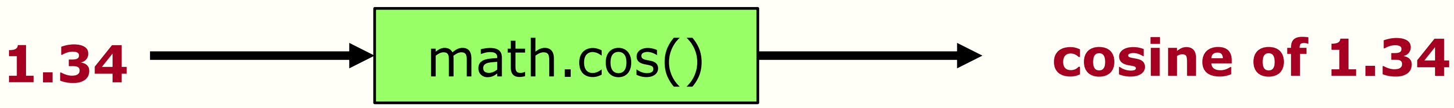
- We talked about functions in Week 1

```
import math

x = math.cos(math.pi/4)

```

- We will show you how to write your own functions

Your first function

- You know that when you use the function math.cos(), you input a value and get an output



- You will now write a function which squares the input value and then outputs it

my_square()

- Open the file my_square_prelim.py that comes with this week's lecture
- Type in Lines 12-14 as shown below
  - Don't forget the : at the end of Line 12
  - The indentation in Lines 13-14 is important
- And then run the program

```
12 def my_square(x):
13     y = x ** 2
14     return y
15
16 a = 5
17 b = my_square(a)
18 print('The value of b is',b)
```

Anatomy of a function

- Line 12:
  - **def** means you want to define a function
  - The name of the function is my_square
  - x is the identifier you give to the function input
- Lines 13 and 14 are indented relative to **def** so they belong to the function definition
- Lines 16 and 17 are *not* indented, so they are *not* part of the function

```
12 def my_square(x):
13     y = x ** 2
14     return y
15
16 a = 5
17 b = my_square(a)
18 print('The value of b is',b)
```

Mechanics of function evaluation (1)

- Line 17: The function my_square is called
  - Terminology: Calling a function means executing the code inside a function
- Because the variable **a** has the value of 5, the identifier x in the function is assigned the value of 5
- The code inside the function is executed sequentially
- Line 13: the identifier y is assigned the value of 25

```
12 def my_square(x):
13     y = x ** 2
14     return y
15
16 a = 5
17 b = my_square(a)
18 print('The value of b is',b)
```

Mechanics of function evaluation (2)

- **return** y in Line 14 means the value of y (which is 25) is to be put at <u>the place where the function is called</u>
- The right-hand-side of Line 17 is now 25
- b is then assigned the value of 25

```
12 def my_square(x):
13     y = x ** 2
14     return y
15 
16 a = 5
17 b = my_square(a)
18 print('The value of b is',b)
```

Multiple inputs

- Code in my_power.py
- You can have multiple inputs to a function
  - For example, the function my_power has two inputs (Line 12)
- When the function my_power is called in Lines 15 and 17, there are 2 values inside the parentheses separated by a comma

```
12 def my_power(x,n):
13     return x ** n
14
15 print('The value of my_power(5,2) is',my_power(5,2))
16
17 print('The value of my_power(2,5) is',my_power(2,5))
```

Orderly assignment

```
def my_power(x,n):
    return x ** n

print('The value of my_power(5,2) is',my_power(5,2))

print('The value of my_power(2,5) is',my_power(2,5))
```

$x \leftarrow 5$

$n \leftarrow 2$

$x \leftarrow 2$

$n \leftarrow 5$

Local scope

- The code is in local.py
- Note that there is a variable y in the function and there is also a variable y outside the function
- Are they the same?

```
def my_power(x,n):
    y = x ** n
    return y


y = 4
z = my_power(y,2)


print('y = ' ,y)
print('z = ' ,z)
```

We will copy the code to the Python tutor website which allows us to visualise the execution of the code

http://pythontutor.com/visualize.html

Local variable scope

- The variables in the function are stored in a separate memory space
  - This applies to data types int, float, str, bool
  - But not for all data types, will tell you more later
- We say the scope of the variable is **local** to the function

```
def my_power(x,n):
    y = x ** n
    return y

y = 4
z = my_power(y,2)

print('y = ' ,y)
print('z = ' ,z)
```

Memory space for my_power	
x	4
n	2
y	16

Base memory space	
y	4

Multiple outputs

- Code in my_power3.py

```
def my_power3(x,n1,n2,n3):
    y1 = x ** n1
    y2 = x ** n2
    y3 = x ** n3
    return y1, y2, y3


a1, a2, a3 = my_power3(5,2,3,4)

print('The values of a1',a1)
print('The values of a2',a2)
print('The values of a3',a3)
```

$x \leftarrow 5$

$n1 \leftarrow 2$

$n2 \leftarrow 3$

$n3 \leftarrow 4$

Functions can call other functions

- The code is in my_power3_improved.py
- A function can call other functions

```python
def my_power3(x,n1,n2,n3):
    y1 = x ** n1
    y2 = x ** n2
    y3 = x ** n3
    return y1, y2, y3

```

```python
def my_power3(x,n1,n2,n3):
    y1 = my_power(x,n1)
    y2 = my_power(x,n2)
    y3 = my_power(x,n3)
    return y1, y2, y3

def my_power(x,n):
    return x ** n
```

Function must be defined before they can be called

- Python expects that you define the functions before they are called
- The following code will **not** work because the function my_square is called in Line 13 but its definition is only found later in Line 16

```
12 a = 5
13 b = my_square(a)
14 print('The value of b is',b)
15
16 def my_square(x):
17     y = x ** 2
18     return y
```

DOESN'T WORK

Function must be defined before they can be called (cont'd)

- Consider the code my_power3_improved.py where the function my_power3() calls the function my_power()
- The function my_power3() is called the first time in Line 22. So my_power3() and the function it calls must all be defined before this line.
- However, my_power3() and the functions it calls can appear in any order.

```
11 # Define the functions
12 def my_power3(x,n1,n2,n3):
13     y1 = my_power(x,n1)
14     y2 = my_power(x,n2)
15     y3 = my_power(x,n3)
16     return y1, y2, y3
17
18 def my_power(x,n):
19     return x ** n
20
21 # Calling the funciton my_power3
22 a1, a2, a3 = my_power3(5,2,3,4)
```

Can exchange the order but both must be defined before my_power3() is called

This week's topics

- Selection structure
- Functions
- List
- Plotting

List

- You have come across a number of data types:
  - int, float, str, bool, etc.

- We will now introduce a new data type called list

- A list consists of a sequence of objects enclosed in [ ] and separated by commas

```
x = [1,-5,7.2,10,-178]
y = ['true','or','dare']

a = 5
b = -6
z = ['hotchpotch',a,b,a*b]
```

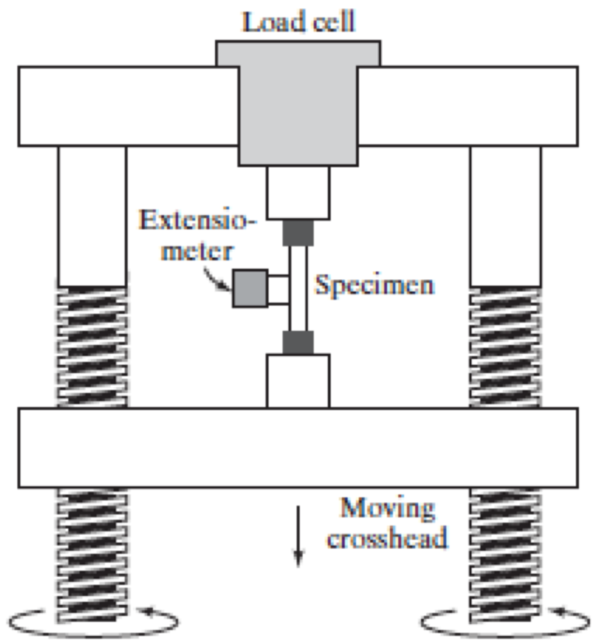
List is useful because you can use them with loops

Line continuation

- Sometimes you have a long line of code, it is best that split it into multiple lines so that you don't have to scroll to the right to read
- Python uses two methods to say that code typed in multiple lines of code is in fact one line of code
  - Implicit continuation with brackets (), [], {}
  - Explicit continuation with \
- Demo code in continuation.py

Tensile testing machine

- To understand how materials behave under tensile force
- Pull the specimen and measure its length



Source: Holly Moore, Matlab for Engineers. p.197

Data from a test

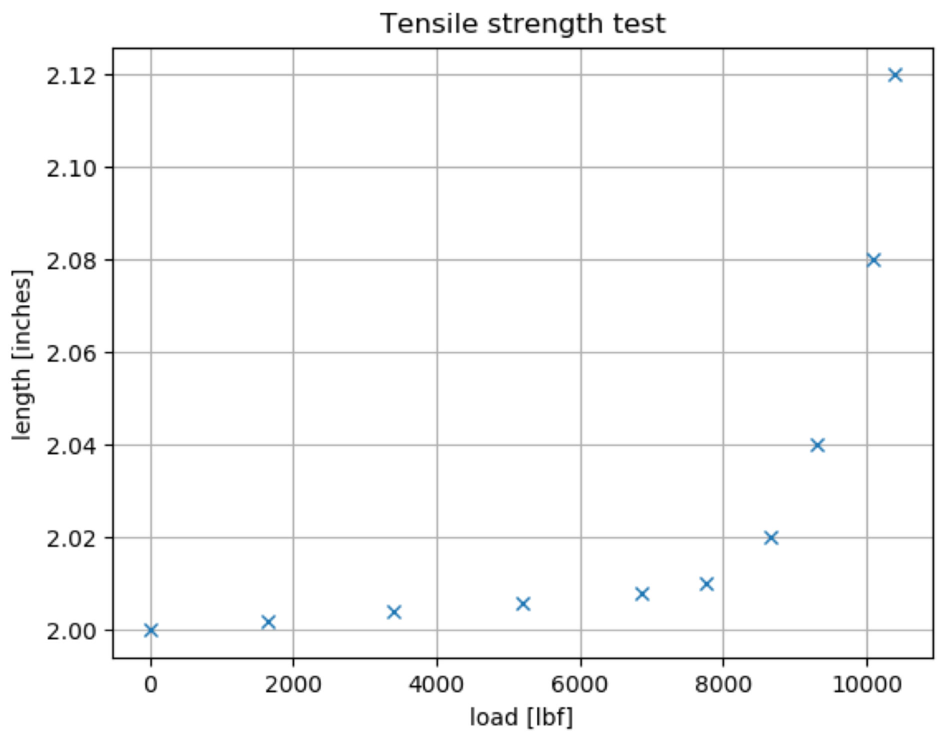
Load [lbf]	Length [inches]
0	2.000
1650	2.002
3400	2.004
5200	2.006
6850	2.008
7750	2.010
8650	2.020
9300	2.040
10100	2.080
10400	2.120

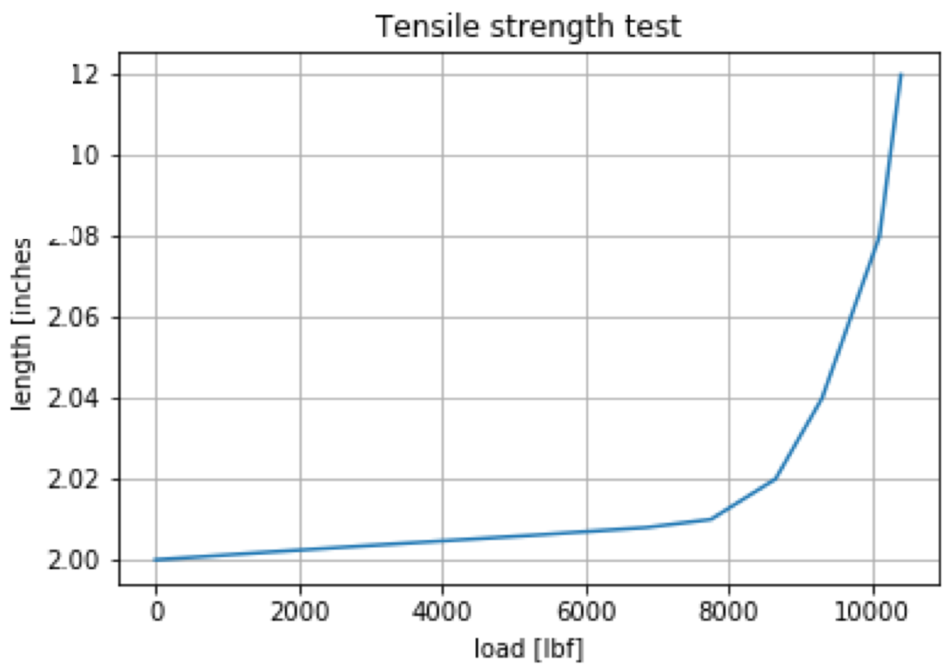
- Make two lists
  - One for load
  - The other for length
- Plot load on the horizontal axis and length on the vertical axis

Source: Holly Moore, Matlab for Engineers. p.197

Plotting graph

- The code is in plot_demo.py





Code for graph plotting

Import library

Short form

Lines 11-15

Put the data in 2 lists

```
9  import matplotlib.pyplot as plt
10
11 load = [0, 1650, 3400, 5200, 6850, 7750, 8650,
12         9300, 10100, 10400]
13
14 length = [2.000, 2.002, 2.004, 2.006, 2.008, 2.010, 2.020,
15           2.040, 2.080, 2.120]
16
17 fig1 = plt.figure()          # create a new figure
18 plt.plot(load,length,'x')    # Plot each point with a cross
19 # plt.plot(load,length)        # plot(data in x-axis, data in y-axis)
20 # The above command will join the points with line
21 plt.xlabel('load [lbf]')     # label for
22 plt.ylabel('length [inches]')    # label
23 plt.title('Tensile strength test') # ti
24 plt.grid()  # display the grid
25 plt.show()  # to display the graph
26 fig1.savefig('tensil_test.png') # save the graph as a PNG file
27 fig1.savefig('tensil_test.pdf') # save the graph as a PDF file
```

Lines 17-27

Plotting graphs

matplotlib

- matplotlib is a large library with many functions
- You can do plots of many different styles
  - Pie chart, histogram, log-log, log-linear, 3D and even animation
- And also to customise them in many ways
- We will only show you the basic plot types
- The library is well documented and its website has many examples
  - https://matplotlib.org

Summary

- Control structures
  - if, if/else, if/elif/else
- Boolean
  - Relational operators: >, <, ==, !=, <=, >=
  - Boolean operators: and, or, not

- Program development
  - Incremental development
  - Planning before writing
  - Flowcharts and pseudocode

- Functions

- List

- Plotting

End

Week 2: Selection Structures, Functions, List, Plotting